PEMBUATAN APLIKASI *PREDICTIVE TEXT* MENGGUNAKAN METODE *N-GRAM-BASED*

Sendy Andrian Sugianto[1], Liliana[2], Silvia Rostianingsih[3]
Program Studi Teknik Informatika Fakultas Teknologi Industri Universitas Kristen Petra
Jl. Siwalankerto 121 – 131 Surabaya 60236
Telp. (031) – 2983455, Fax. (031) – 8417658
E-mail: sendy_andrian@yahoo.com[1], lilian@petra.ac.id[2], silvia@petra.ac.id[3]

ABSTRAK

Penggunaan teknologi dalam pengetikan sudah sangat berkembang. Bermulai dari mesin ketik, lalu mengalami perubahan kemajuan teknologi menjadi komputer, dan yang paling marak adalah telepon genggam. Semua itu adalah media pengetikkan yang masih berkembang sampai saat ini. Teknologi tidak hanya berkembang pada media pengetikkannya saja, tetapi juga berkembang pada sistem proses kata. Sistem-sistem prediksi pun banyak bermunculan untuk membantu mempercepat proses pengetikkan.

Oleh karena itu, pada skripsi ini dilakukan pembuatan aplikasi untuk melakukan prediksi kata dengan menggunakan *n-gram* sebagai metode dasar melakukan proses prediksi. Proses dimulai dengan memecah kata per kata dan mengelompokkannya sesuai dengan *language model*. Kemudian, dilakukan proses *scoring* untuk menentukan kata mana yang sesuai untuk menjadi pilihan prediksi kata.

Hasil pengujian menunjukkan bahwa metode *n-gram* sebagai metode dasar dalam proses prediksi sangatlah membantu pemilahan kata, sehingga proses prediksi menjadi lebih efektif, mampu menghasilkan prediksi efektif di atas 20% dari total prediksi yang terjadi. *Keystroke saving* yang dapat dihasilkan dapat mencapai 50% tergantung dari data *training* yang digunakan. Selain dari pada metode n-gram sendiri, pengaturan bobot untuk masing-masing *score* kata juga sangat mempengaruhi proses prediksi kata.

Kata Kunci
Predictive Text, *N-Gram, Keystroke Saving*.

ABSTRACT

The Typing technology has been highly developed. From the used of typewriters, then it changes into the computer technology, and the most prevalent is mobile device. They are the typing technology that still envolving until today. Technology is not only about the typing method, but also thrives on word processing systems. There are many words prediction that appear to help human speed up their typing.

Therefore, this thesis making an application to perform word prediction using n-gram based method to do the prediction process. The process begins by parsing the words and grouping them according to the language model. Then, scoring process determine which words are the appropriate choice of word prediction.

The test result says that n-gram based method is very helpful in the process of sorting word prediction, so that prediction process became more effective, and able to produce effective prediction above 20% of the total prediction. Keystroke savings that can be generated can reach 50% depending on the training data are used. Other than n-gram based method, setting of the weights for each scoring method also affect the process of word prediction.

Keywords
Predictive Text, N-Gram, Keystroke Saving.

1. PENDAHULUAN

Perkembangan penggunaan teknologi dalam pengetikan sudah sangat berkembang. Hal ini terbukti dari pengetikan untuk sebuah dokumen, seperti surat, tugas sekolah, laporan dalam perusahaan, yang dulunya menggunakan mesin ketik dan sekarang sudah menggunakan komputer. Pengetikan yang dilakukan pada komputer berbeda dengan mesin ketik yang hasilnya langsung dicetak pada kertas. Saat mengetik pada komputer, hasil ketikan tidak langsung dicetak pada sebuah kertas, namun akan disimpan dalam sebuah file yang bisa diubah isinya sesuai dengan keinginan pengetik.

Dalam proses pengetikan, sering ditemukan salah pengetikan kata di dalam sebuah dokumen baik dokumen yang berupa *file* ataupun dokumen yang sudah dicetak. Penulisan yang salah dalam dokumen penting pun sangatlah dihindari, karena apabila terdapat kesalahan pengetikan dalam sebuah dokumen penting, bisa saja dokumen tersebut menjadi tidak valid atau tidak diakui kebenarannya.

Selain kesalahan penulisan kata dalam pengetikan, program editor atau program yang digunakan untuk mengetik dibutuhkan untuk bisa mengetahui kata-kata apa saja yang sering digunakan. Dengan mengetahui kata-kata yang sering digunakan oleh pengetik, hal ini dapat mengurangi kemungkinan terjadinya kesalahan dalam pengetikan kata dan menjadikan proses pengetikan lebih efektif karena proses pengetikan menjadi lebih cepat. Jadi, dengan hanya mengetikkan beberapa karakter awal dari kata tersebut, kata-kata yang dimaksud akan muncul dengan sendirinya.

2. PREDICTIVE TEXT

Predictive Text adalah sebuah fitur pada pengetikan dimana untuk menuliskan satu kata, *user* cukup mengetikkan satu atau lebih huruf awal dari kata yang dimaksud. Hal ini bertujuan untuk mengurangi *keystroke* atau tombol yang digunakan dalam pengetikan untuk memproduksi sebuah pesan teks. *User* dibantu untuk mempercepat proses pengetikan, dimana *user* tidak perlu mengetikkan satu per satu karakter dari *text* target. Contohnya, saat mengetikkan huruf "a" yang diikuti huruf "b", maka secara otomatis akan tersusun menjadi beberapa kata yang muncul di bawahnya yang nantinya dipilih, seperti "aba", "abad", "abadi", "abdi", dan lain sebagainya.

Skenario untuk *predictive text* adalah dengan mengandaikan bahwa pemrediksi bekerja pada beberapa segmen yang dirancang

dari sumber *text*, dan menguraikan kata-kata dari kiri ke kanan. Setiap karakter *text* target diketik, prediksi yang diusulkan untuk kata akan ditampilkan. Jika prediksi kata benar, *user* dapat menerimanya tanpa harus mengetikkan tiap karakter dari *text* target. *User* dapat menerima usulan prediksi dengan menekan tombol tertentu sebagai tombol konfirmasi dari *user*, dan *user* dapat langsung mengetikkan kata berikutnya.

Saat ini, *predictive text* telah banyak diaplikasikan pada perangkat komunikasi yang membutuhkan *input text*, seperti komputer, *personal digital assistant* (PDA), dan telepon selular. "*Predictive text* yang paling populer dan banyak dijumpai adalah T9 pada *mobile device*, dan *Letterwise* yang digunakan oleh alat olah kata pada komputer". Metode T9 adalah *predictive text* yang digunakan pada *mobile device* yang menggunakan 9 *keys* atau *numeric keypad*. Metode ini didasarkan pada tata letak tombol yang sama sebagai metode *multi-press*, tetapi masing-masing tombol ditekan hanya sekali. T9 membandingkan kemungkinan kata ke *database* linguistik untuk "menebak" kata dimaksudkan [1]. Misalnya, untuk memasukkan "*the*", *user* memasukkan urutan tombol 8-4-3-0, '0' berfungsi sebagai kata pembatas dan mengakhiri keambiguan satu kunci sebelumnya. Setelah itu akan muncul semua kata dari *dictionary* yang sesuai dengan urutan tombol beserta frekuensi penggunaan masing-masing kata tersebut. Sedangkan *Letterwise* adalah sistem *predictive text* lain yang tidak bergantung pada *dictionary*, yang memungkinan bagi *user* untuk bebas mengetikkan kata-kata, namun mempunyai efisiensi yang sangat tinggi. Sistem ini sangat mudah digunakan dengan tanpa keterbatasan kamus, dan instruksi manual hanya satu kalimat : "*Hit the key with the letter you want, if it doesn't come up, hit Next until it does.*".

Metode lain yang dapat digunakan untuk mengolah *predictive text*, salah satunya yakni *n-gram*. Probabilitas pada *n-gram* tidak sama dengan probabilitas kata P(w), yang menghitung kata w terlepas dari kata di sebelahnya. Model n-gram menghitung probabilitas bersyarat P (w | g), probabilitas untuk sebuah kata w dari urutan kata sebelumnya yakni kata g [2]. Dalam istilah lain, yaitu memprediksi kata berikutnya berdasarkan kata sebelumnya kata n-1. Sebagai contoh, untuk menemukan probabilitas bersyarat P (hitam | kucing) terdiri dari menghitung probabilitas dari seluruh urutan "kucing hitam". Dengan kata lain, untuk kata "hitam", probabilitas yang akan dihitung adalah probabilitas kata setelah kata "kucing".

Selain metode n-gram sendiri, akan digabungkan juga beberapa fungsi *scoring* yang mendukung *predictive text*, seperti *language model* dan *semantic affinity*. *Language model* didasarkan pada urutan kata dan kata yang paling sering digunakan dalam *input*-an teks, sedangkan *semantic affinity* didasarkan pada kemungkinan kata tersebut muncul bersama dalam sebuah kalimat [3].

2.1 N-Gram

Pada dasarnya, model n-gram adalah model probabilistik yang awalnya dirancang oleh ahli matematika dari Rusia pada awal abad ke-20 dan kemudian dikembangkan untuk memprediksi *item* berikutnya dalam urutan *item*. *Item* bisa berupa huruf / karakter, kata, atau yang lain sesuai dengan aplikasi. Salah satunya, model *n-gram* yang berbasis kata digunakan untuk memprediksi kata berikutnya dalam urutan kata tertentu. Dalam arti bahwa sebuah *n-gram* hanyalah sebuah wadah kumpulan kata dengan masing-masing memiliki panjang n kata. Sebagai contoh, sebuah *n-gram* ukuran 1 disebut sebagai *unigram*; ukuran 2 sebagai "*bigram*"; ukuran 3 sebagai "*trigram*", dan seterusnya.

Pada pembangkitan karakter, ***N-gram*** terdiri dari ***substring*** sepanjang ***n*** karakter dari sebuah ***string*** dalam definisi lain ***n-gram*** adalah potongan sejumlah ***n*** karakter dari sebuah ***string***. Metode ***n-gram*** ini digunakan untuk mengambil potongan-potongan karakter huruf sejumlah ***n*** dari sebuah kata yang secara kontinuitas dibaca dari teks sumber hingga akhir dari dokumen. Sebagai contoh : kata "*TEXT*" dapat diuraikan ke dalam beberapa ***n-gram*** berikut:

uni-gram : T, E, X, T
bi-gram : TE, EX, XT
tri-gram :, TEX, EXT
quad-gram : *TEXT*, EXT_
dan seterusnya.

Sedangkan pada pembangkitan kata, metode ***n-gram*** ini digunakan untuk mengambil potongan kata sejumlah ***n*** dari sebuah rangkaian kata (kalimat, paragraf, bacaan) yang secara kontinuitas dibaca dari teks sumber hingga akhir dari dokumen. Sebagai contoh : kalimat "*saya dapat melihat cahaya itu.*" dapat diuraikan ke dalam beberapa ***n-gram*** berikut :

uni-gram : saya, dapat, melihat, cahaya, itu
bi-gram : saya dapat, dapat melihat, itu ada
tri-gram : saya dapat melihat, dapat melihat dia
dan seterusnya.

Salah-satu keunggulan menggunakan ***n-gram*** dan bukan suatu kata utuh secara keseluruhan adalah bahwa ***n-gram*** tidak terlalu sensitif terhadap kesalahan penulisan yang terdapat pada suatu dokumen [4].

2.2 Language Model

Language Model adalah metode *input* yang berdasarkan frekuensi kata dan tidak peka terhadap konteks [5]. Hal ini berarti bahwa kata-kata sebelumnya dalam sebuah kalimat tidak dianggap saat menyortir alternatif kata. Untuk meningkatkan konteks dan menghasilkan disambiguasi yang baik dibutuhkan cara yang lebih baik. Berdasarkan teori statistik muncul gagasan *Maximum Likelihood Estimate* (MLE) yang dapat digunakan untuk membuat estimasi tentang bagaimana kemungkinan urutan kata-kata tersebut. Estimasi didasarkan pada seberapa sering urutan kata-kata tersebut muncul di dalam teks *input*-an.

Statistik dikumpulkan dari korpus. Contoh *n-gram* dari bahasa swedia yang ditampilkan pada Tabel 1.

Tabel 1. (a) *bigram*, (b) *trigram*

w_{i-2}	w_{i-1}	w_i	#
allt	det	där	10
,	och	där	10
på	den	där	8
det	var	där	7
,	den	där	7
här	och	där	7

(a)

w_{i-1}	w_i	#
,	där	387
den	där	104
och	där	83
det	där	77
de	där	24
så	där	21

(b)

N-gram bersifat eksponensial dalam penggunaan memori. Hal tersebut akan mulai berpengaruh saat korpus sudah berisi 10,000 kata yang berbeda. Untuk menyimpan semua kombinasi yang mungkin diperlukan matriks ukuran $10{,}000^2$ untuk *bigram*, $10{,}000^3$ untuk *trigram* dan sebagainya sampai dengan $10{,}000^i$ untuk *n-gram* ukuran i. Hal ini berarti bahwa model yang

didasarkan pada *n-gram* yang lebih besar dari 3, mustahil untuk menjaga *memory* selama *runtime* [5].
Sebuah interpolasi linier dari *n-gram* yang berbeda menjadi :

$$P(w_i|w_{i-n+1}\dots w_{i-1}) = \lambda_i P(w_i) + \dots + \lambda_n P(w_i|w_{i-n+1}\dots w_{i-1}) \quad (1)$$

dimana λ_i adalah *weights* t untuk spesifik *n-gram* dan

$$0 \leq \lambda\_i \leq 1 \quad (2)$$

dan

$$\sum \lambda_i = 1 \quad (3)$$

sehingga fungsi skor *Language Model* dapat digabungkan dengan fungsi skor yang lain, dan bobot / *weights* ditentukan dengan mengiterasi subset dari kombinasi yang mungkin [5]. Hal ini dilakukan dalam langkah-langkah tambahan untuk menemukan *weights* supaya hasil dari *Key Stroke Per Character* (KSPC) bisa menjadi paling rendah untuk satu set pembangunan. *Weights* bisa mengakibatkan KSPC menjadi paling rendah saat fungsi skor *Language Model* digunakan dalam kombinasi dengan fungsi skor lainnya.
Berdasarkan analisis penggunaan model *n-gram*, *language model* akan kurang efektif apabila menggunakan model *n-gram* dengan nilai n yang besar. Selain sangat mustahil menjaga *memory* selama *runtime*, frekuensi keluar dari hasil *language model* yang didapat saat n bernilai besar, juga akan lebih jarang.
Hasil analisis penggunaan model *n-gram* dalam *language model*, menunjukkan bahwa semakin besar nilai n yang digunakan dalam *n-gram* berbanding terbalik dengan jumlah frekuensi keluar yang didapat, yaitu semakin kecil atau lebih jarang keluar. Penggunaan model *bi-gram* dan *tri-gram* untuk *language model* masih memungkinkan, karena hasil dari jumlah frekuensi keluar pada suku *n-gram*-nya masih cukup besar dan datanya masih valid apabila diproses lebih lanjut.

2.3 Semantic Affinity

Dalam kalimat yang panjang, sangat memungkinkan untuk terjadi hubungan semantik antara kata yang satu dengan yang lain. Hubungan semantik tersebut mungkin saja terjadi dengan jarak yang jauh tergantung dari *n-gram* model yang digunakan, sehingga *language model* melewatkannya [5]. Dengan menggunakan *n-gram* yang panjangnya tidak lebih dari tiga, model hanya membutuhkan kata-kata tersebut berada dalam sebuah kelompok yang masing-masing maksimal hanya berisi tiga kata di dalam korpus.
Keterkaitan semantik antara dua kata untuk mengukur hubungan dalam kalimat [6], terdefinisi sebagai berikut :

$$SemR(w_i, w_j) = \frac{C(w_i, w_j)}{C(w_i)C(w_j)} \quad (4)$$

dimana $C(w_i,w_j)$ adalah jumlah berapa kali kata-kata w_i dan w_j terjadi dalam sebuah kalimat di dalam korpus, dan $C(w_i)$ adalah jumlah kata w_i di dalam korpus. Hubungan simetrisnya adalah :

$$C(w_i, w_j) = C(w_j, w_i) \quad (5)$$

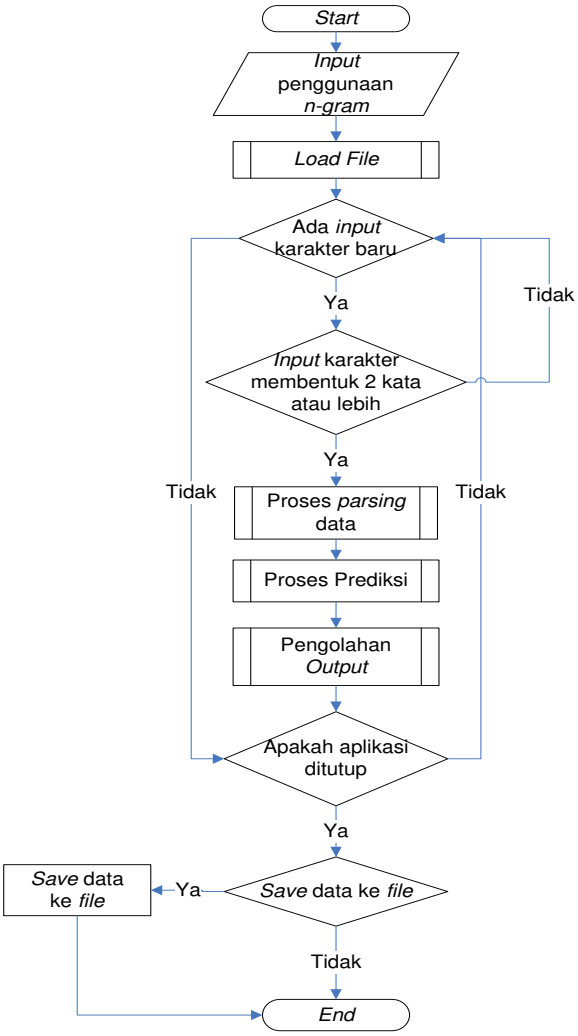
Afinitas semantiknya diperkirakan dari sebuah kata w, didefinisikan sebagai berikut :

$$SemA(w|H) = \sum_{w_j \in H} SemR(w, w_j) \quad (6)$$

dimana H adalah konteks kata w. Dalam kasus ini, H mengandung kata-kata dari sebelah kiri kata saat ini. Penggunakan konteks perubahan ukuran dan menemukan sebuah *keystroke saving rate* (KSR) antara 64.86% dan 65.80% tergantung pada ukuran konteks, dibandingkan dengan 59% dengan hanya menggunakan kata n-gram [6]. KSR adalah ukuran yang menentukan berapa banyak *keystroke* yang disimpan *user* dengan metode *input* tertentu dibandingkan dengan metode dasar.

3. DESAIN SISTEM

Dalam melakukan *predictive text*, ada beberapa langkah yang harus dilakukan. Pertama sistem mengkonfirmasi metode *n-gram* yang digunakan pada *language model*, lalu membuka atau *load file* kata-kata sesuai metode *n-gram* yang dipilih. Kemudian membaca *input* karakter dari *user* dan melakukan *parsing* data. Setelah itu, sistem melakukan *searching* dan *scoring* kata dari *file* untuk menghasilkan *predictive text*. Terakhir, sistem memberikan usulan kata yang menjadi *predictive text* kepada *user*. Rancangan sistem kerja aplikasi secara garis besar ditunjukkan pada Gambar 1.



Gambar 1. Desain Sistem Kerja *Predictive Text*

Saat awal menjalankan aplikasi, sistem perlu konfirmasi dari *user* apakah proses *predictive text* dilakukan secara *bi-gram* atau *tri-gram*. Selanjutnya, sistem perlu mengambil data dari *file*, dan data yang diambil pada *file* disesuaikan juga dengan pilihan *user* apakah *bi-gram* atau *tri-gram*. Setelah itu aplikasi melakukan *looping* dimana sistem mengecek apakah terdapat *input* karakter baru dari *user*. Apabila terdapat *input* baru, maka sistem langsung melakukan proses berikutnya, yakni proses *parsing*, proses prediksi, dan pengolahan output. Saat sistem tidak menerima *input* baru, sistem mengecek apakah aplikasi ditutup atau tidak. Apabila aplikasi ditutup, sistem menawarkan penyimpanan data dari kata-kata yang baru diketik untuk disimpan pada *file*.

3.1 Parsing Data

Saat ditemukan adanya *input* karakter baru dari *user*, sistem langsung melakukan *parsing* data terhadap kata-kata yang terbentuk hasil dari *input* karakter. *Parsing* data mulai dilakukan ketika terdapat 2 kata atau lebih, bergantung pada metode *bi-gram* atau *tri-gram*. Apabila kurang dari itu, sistem belum memulai proses *parsing* data dan kembali mengecek input karakter dari *user*. Proses *Parsing* Data adalah proses untuk membagi kata-kata menjadi rangkaian kata-kata sepanjang n, dimana n disesuaikan dengan metode *n-gram* yang telah dipilih. Untuk proses *parsing* data sendiri, kata-kata yang di-*parsing* adalah kata-kata dari dokumen baru / dokumen yang sedang aktif yang telah terdapat *input*-an karakter baru dari *user* dan sudah membentuk kata.

Awalnya kata-kata tersebut dibersihkan dulu dari tanda baca atau tanda-tanda lain yang tidak dibutuhkan pada proses *cleaning string*. Maksudnya, tanda baca atau tanda-tanda lain yang tidak dibutuhkan dihilangkan atau dihapus. Selanjutnya, kata dipecah berdasarkan spasi menjadi kumpulan kata yang terdiri dari masing-masing satu kata. Setelah itu, kata kembali dikelompokkan sesuai dengan metode *n-gram* yang dipilih, *bi-gram* terdiri dari kelompok 2 kata sedangkan *tri-gram* terdiri dari kelompok 3 kata.

3.2 Proses Prediksi

Proses ini menyangkut tentang penentuan kata mana saja yang nantinya diusulkan menjadi pilihan kata *predictive text*. Pada proses ini ada dua langkah yang penting, yang pertama adalah pencarian kata mana saja yang berkompeten untuk menjadi pilihan kata *predictive text* dari *file* (proses *searching*), dan yang kedua adalah proses penilaian dari tiap-tiap kata yang sudah dipilih untuk menjadi pilihan *predictive text* (proses *scoring*).

Sebelum memulai kedua proses tersebut, sistem perlu tahu data atau kata *input* apa yang akan dicari dan diproses. Setelah mendapatkan data *input*, proses mengolah data input tersebut pada proses *searching*. Proses *searching* mencari kata-kata yang berpotensi menjadi usulan kata *predictive text* pada *file*.

Dalam proses ini, kata-kata yang berkompeten diolah untuk menjadi pilihan *predictive text*. Ada 3 penilaian yang dilakukan, yang pertama adalah penilaian berdasarkan frekuensi kata pernah keluar, baik dalam dokumen yang sedang diketik user maupun dalam dokumen-dokumen sebelumnya (*Frequency Scoring*). Kedua adalah penilaian berdasarkan hubungan semantik kata (*Semantic Scoring*). Sedangkan yang terakhir adalah penilaian berdasarkan kesamaan suku *n-gram* yang dimiliki oleh kata kandidat dibandingkan dengan suku-suku *n-gram* dari kata *input*.

4. IMPLEMENTASI DAN PENGUJIAN SISTEM

Aplikasi diuji dengan beberapa pengujian untuk mengetahui sejauh mana aplikasi layak dipergunakan. Pengujian dilakukan pada dua *language model*, yakni *Bigram* dan *Trigram*. Pada awal pengujian, pertama-tama di-*input*-kan sebuah teks terlebih dahulu, lalu disimpan pada *file*, setelah itu dicoba untuk mengetikkan kembali teks tersebut dari karakter awal, dan dinilai sesuai dengan pengujian. Pengujian yang dilakukan antara lain menguji bobot dari tiap metode *scoring*-nya, yakni *Frequency Scoring* (FS), *Semantic Scoring* (SS), dan *N-gram Scoring* (NS) berdasarkan formula yang sudah disediakan, lalu juga pengecekan terhadap prediksi efektif yang terjadi, dan juga menghitung *Keystroke Saving* (KS) yang dihasilkan dalam mengetikkan beberapa artikel baik cerpen, berita ataupun yang lain.

Pengujian bobot pada tiap metode *scoring* bertujuan untuk mengetahui kombinasi bobot yang efektif dalam melakukan *scoring* pada pilihan kata prediksi. Formula bobot *scoring* yang diujikan pada aplikasi ditunjukkan pada Tabel 2. prediksi kata yang diujikan pada aplikasi. Selain itu, pengujian juga dilakukan pada hasil prediksi. Pengujian dilakukan dengan menghitung seberapa banyak prediksi efektif yang didapat dari sebuah pengetikkan artikel. *Score* Prediksi Efektif (SPE) didapat dari jumlah prediksi efektif yang terjadi dibandingkan dengan jumlah total prediksi yang terjadi.

Tabel 2. Formula Pengujian Scoring

Formula	Weight (%)		
	Frequency Score (FS)	Semantic Score (SS)	Ngram Score (NS)
1	100	0	0
2	50	50	0
3	50	0	50
4	34	33	33

Pengujian dengan menghitung *keystroke saving* adalah untuk menghitung seberapa banyak karakter yang bisa dihemat untuk menghasilkan sebuah teks tertentu. Besar nilai *keystroke saving* didapat dari jumlah selisih karakter yang tidak di-*input*-kan untuk menyelesaikan sebuah teks, dibandingkan dengan jumlah total karakter teks tersebut. Semakin sedikit karakter yang di-*input*-kan, berarti semakin besar pula *keystroke saving* yang dihasilkan. Saat prediksi kata sudah mulai bekerja pada aplikasi, sebuah kata dapat diselesaikan tanpa harus mengetikkan satu persatu karakter dari kata tersebut. Hal ini terjadi karena *user* cukup mengetikkan hanya beberapa karakter awal dari kata yang dituju setelah muncul prediksi kata yang sesuai *user* hanya perlu menekan tombol khusus untuk menyelesaikan kata tersebut.

Pengujian dilakukan pada dua *language model*, yakni *Bigram* dan *Trigram*. Langkah awal pengujian ini sama dengan langkah awal pada pengujian *keystroke saving*. Pertama-tama di-*input*-kan sebuah teks terlebih dahulu, lalu disimpan pada *file*, setelah itu dicoba untuk mengetikkan kembali teks tersebut dari karakter awal, dan dinilai sesuai dengan pengujian.

4.1 Hasil Pengujian dengan Data Training dan Data Input yang sama

Dari hasil perhitungan *score* pengujian, didapat hasil dari tiap formula pengujian *scoring*, yang telah diaplikasikan pada kedua *language model*, yakni *bigram* dan *trigram*. Data *input* yang digunakan pada pengujian ini sama dengan data *training*-nya.

Hasil pengujian yang disorot adalah *Keystroke Saving* (KS) dan *Score* Prediksi Efektif (SPE). Tabel 3 menunjukkan hasil rata-rata nilai yang dihasilkan oleh tiap formula pada *Bigram*.

Tabel 3. Hasil Penghitungan Rata-Rata Nilai Tiap Formula Pada *Bigram*

Bigram		
Formula	Nilai Rata-Rata	
	KS (%)	SPE (%)
1 (100% FS)	49,32	58,74
2 (50% FS, 50% SS)	46,792	52,00
3 (50% FS, 50% NS)	**54,261**	**70,55**
4 (34% FS, 33% SS, 33% NS)	53,355	68,39

Berdasarkan hasil rata-rata nilai yang dihasilkan tiap formula, dapat disimpulkan bahwa penggunaan formula *scoring* yang ketiga dapat menghasilkan *score Keystroke Saving* dan *score* Prediksi Efektif terbesar untuk penggunaan *language model Bigram*. Setelah pengujian dilakukan pada *Bigram*, selanjutnya pengujian dilakukan pada *Trigram*. Tabel 4 menunjukkan hasil rata-rata nilai yang dihasilkan oleh tiap formula pada *Trigram*.

Tabel 4. Hasil Penghitungan Rata-Rata Nilai Tiap Formula Pada *Trigram*

Trigram		
Formula	Nilai Rata-Rata	
	KS (%)	SPE (%)
1 (100% FS)	49,812	60,43
2 (50% FS, 50% SS)	47,806	54,50
3 (50% FS, 50% NS)	**54,462**	**70,99**
4 (34% FS, 33% SS, 33% NS)	54,206	70,58

Ternyata hasil yang didapat dari pengujian pada *Trigram* ini, sama dengan hasil yang didapat dari pengujian pada *Bigram*. Berdasarkan hasil rata-rata nilai yang dihasilkan tiap formula, dapat disimpulkan bahwa penggunaan formula *scoring* yang ketiga dapat menghasilkan *score Keystroke Saving* dan *score* Prediksi Efektif terbesar untuk penggunaan *language model Trigram*.

Secara keseluruhan, hasil dari pengujian dengan data *training* dan data *input* yang sama, didapati bahwa *keystroke saving* yang dihasilkan dapat mencapai 54 persen bergantung pada data *training* yang digunakan. *Keystroke Saving* juga dipengaruhi oleh kombinasi bobot pada tiap metode *scoring* dan juga frekuensi *language model* yang tinggi. Sedangkan untuk prediksi efektif, rata-rata prediksi efektif terjadi di atas 50% dari total prediksi yang terjadi. Hal ini mengacu pada semakin sering aplikasi digunakan oleh *user,* maka semakin tinggi pula hasil dari *keystroke saving* dan frekuensi prediksi efektif yang terjadi.

4.2 Hasil Pengujian dengan Data Training dan Data Input yang berbeda

Pada pengujian ini, data yang di jadikan data *training* pada aplikasi berbeda dengan data input yang diujikan. Jumlah data *training* pada pengujian ini sedikit lebih banyak karena data *training* dikelompokkan berdasarkan genre tiap datanya dan di-*training*-kan secara bersamaan. Sama seperti pengujian sebelumnya, pengujian ini juga diaplikasikan pada kedua *language model*, yakni *bigram* dan *trigram*. Hasil pengujian yang disorot juga sama, yakni *Keystroke Saving* (KS) dan *Score* Prediksi Efektif (SPE). Tabel 5 menunjukkan hasil rata-rata nilai yang dihasilkan oleh tiap formula pada *Bigram*.

Tabel 5. Hasil Penghitungan Rata-Rata Nilai Tiap Formula Pada *Bigram*

Bigram		
Formula	Nilai Rata-Rata	
	KS (%)	SPE (%)
1 (100% FS)	18,12	37,60
2 (50% FS, 50% SS)	16,39	27,05
3 (50% FS, 50% NS)	**18,85**	**41,37**
4 (34% FS, 33% SS, 33% NS)	18,42	38,40

Berdasarkan hasil rata-rata nilai yang dihasilkan tiap formula, dapat disimpulkan bahwa penggunaan formula *scoring* yang ketiga dapat menghasilkan *score Keystroke Saving* dan *score* Prediksi Efektif terbesar untuk penggunaan *language model Bigram*. Setelah pengujian dilakukan pada *Bigram*, selanjutnya pengujian dilakukan pada *Trigram*. Tabel 6 menunjukkan hasil rata-rata nilai yang dihasilkan oleh tiap formula pada *Trigram*.

Tabel 6. Hasil Penghitungan Rata-Rata Nilai Tiap Formula Pada *Trigram*

Trigram		
Formula	Nilai Rata-Rata	
	KS (%)	SPE (%)
1 (100% FS)	18,06	37,70
2 (50% FS, 50% SS)	16,38	26,21
3 (50% FS, 50% NS)	**18,75**	**41,50**
4 (34% FS, 33% SS, 33% NS)	17,77	34,63

Ternyata hasil yang didapat dari pengujian pada *Trigram* ini, sama dengan hasil yang didapat dari pengujian pada *Bigram*. Berdasarkan hasil rata-rata nilai yang dihasilkan tiap formula, dapat disimpulkan bahwa penggunaan formula *scoring* yang ketiga dapat menghasilkan *score Keystroke Saving* dan *score* Prediksi Efektif terbesar untuk penggunaan *language model Trigram*.

Hasil keseluruhan dari pengujian dengan data *training* dan data *input* yang berbeda, didapati bahwa *keystroke saving* yang dihasilkan dapat mencapai 18 persen bergantung pada data *training* yang digunakan dan kombinasi bobot pada tiap metode *scoring*. Sedangkan untuk prediksi efektif yang dihasilkan, nilainya dapat mencapai 41 persen bergantung pada data *training* dan frekuensi penggunaan aplikasi.

4.3 Pengujian *Hardware*

Selain pengujian pada aplikasi / *software*, dilakukan juga pengujian terhadap *hardware* yang digunakan. Ada beberapa *hardware* yang digunakan dan setiap *hardware* tersebut mempunyai spesifikasi yang berbeda satu sama lain. Pengujian akan dilakukan dengan menghitung *timing process* yang dibutuhkan oleh masing-masing *hardware* untuk melakukan prediksi.

Berdasarkan rata-rata *timing process* yang didapat, ternyata *timing process* yang dibutuhkan untuk melakukan prediksi berbeda-beda tergantung dari *hardware* yang digunakan. Bisa dikatakan juga bahwa komponen *hardware* yang berpengaruh pada proses

prediksi ini adalah *processor* dan memori dari *hardware* tersebut. Hal ini terjadi karena aplikasi lebih membutuhkan kinerja dan kapasitas dari memori dan *processor* itu sendiri untuk berproses.
Selain dari faktor spesifikasi *hardware*, *timing process* juga dipengaruhi oleh aplikasi-aplikasi lain yang sedang aktif / jalan pada *hardware*. Apabila banyak aplikasi lain yang sedang dijalankan, maka secara otomatis kinerja untuk aplikasi juga terbagi-bagi. Hal tersebut menyebabkan kemampuan aplikasi dalam melakukan prediksi juga terpengaruh dan menghasilkan *timing process* yang berbeda.

5. DAFTAR PUSTAKA

[1] Silfverberg, M., MacKenzie, I. S., & Korhonen, P. (2000). Predicting text entry speed on mobile phones. *Proceedings of the ACM Conference on Human Factors in Computing Systems - CHI 2000*, pp. 9-16. New York: ACM.

[2] Bassil, Yousef. (2012). Parallel Spell Checking Algorithm Based on Yahoo! N-Grams Dataset. Retrieved January 12, 2013, from http://www.google. com/url?sa=t&rct=j&q=yousef %20bassil%20parallel%20spell-checking&so urce=web&cd= 1&cad=rja&ved=0CCwQFjAA&url=http%3A%2F%2Farxiv. org%2Fpdf%2F1204.0184&ei=27bKUcDxDIqckAW_nYHg Bg&usg=AFQjCNH9M97lUO5aJj4tr0nHKHdv2FR1ag&bvm =bv.48340889,d.dGI

[3] Jorwall, Jakob. & Ganslandt, Sebastian. (2009). Context-Aware Predictive Text Entry for Swedish using Semantics and Syntax. Retrieved January 12, 2013, from http://fileadmin.cs.lth.se/cs/Personal/Pierre_Nugues/memoires /sebastia n _jakob /sebastian_jakob.pdf

[4] Hanafi, Ahmad. (2009). Pengenalan Bahasa Suku Bangsa Indonesia Berbasis Teks Menggunakan Metode *N-gram*. IT TELKOM

[5] Jorwall, Jakob. & Ganslandt, Sebastian. (2009). Context-Aware Predictive Text Entry for Swedish using Semantics and Syntax. Retrieved January 12, 2013, from http://fileadmin.cs.lth.se/cs/Personal/Pierre_Nugues/memoires /sebastia n _jakob /sebastian_jakob.pdf

[6] Li, Jianhua, and Hirst, Graeme. (2005). *Semantic knowledge in word completion. In Assets '05: Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*. Baltimore.